上海建桥学院
SHANGHAI JIAN QIAO UNIVERSITY

6 事件处理机制

万永权

5.6 界面事件

•5.6.1 按键事件

- 在MVC模型中，控制器根据界面事件（UI Event）类型不同，将事件传递给界面控件不同的事件处理函数
  - 按键事件（KeyEvent）将传递给onKey()函数进行处理
  - 触摸事件（TouchEvent）将传递给onTouch()函数进行处理

5.6 界面事件

•5.6.1 按键事件

- Android系统界面事件的传递和处理遵循一的规则
  - 如果界面控件设置了事件监听器，则事件将先传递给事件监听器
  - 如果界面控件没有设置事件监听器，界面事件则会直接传递给界面控件的其他事件处理函数
  - 即使界面控件设置了事件监听器，界面事件也可以再次传递给其他事件处理函数

5.6 界面事件

•5.6.1 按键事件

- Android系统界面事件的传递和处理遵循一的规则
  - 是否继续传递事件给其他处理函数是由事件监听器处理函数的返回值决定的
  - 如果监听器处理函数的返回值为true，表示该事件已经完成处理过程，不需要其他处理函数参与处理过程，这样事件就不会再继续进行传递
  - 如果监听器处理函数的返回值为false，则表示该事件没有完成处理过程，或需要其他处理函数捕获到该事件，事件会被传递给其他的事件处理函数

5.6 界面事件

•5.6.1 按键事件

- 以EditText控件中的按键事件为例，说明Android系统界面事件传递和处理过程，假设EditText控件已经设置了按键事件监听器
  - 当用户按下键盘上的某个按键时，控制器将产生KeyEvent按键事件
  - Android系统会首先判断EditText控件是否设置了按键事件监听器，因为EditText控件已经设置按键事件监听器OnKeyListener，所以按键事件先传递到监听器的事件处理函数onKey()中

5.6 界面事件

•5.6.1 按键事件

- 事件能够继续传递给EditText控件的其他事件处理函数，完全根据onKey()函数的返回值来确定
- 如果onKey()函数返回false，事件将继续传递，这样EditText控件就可以捕获到该事件，将按键的内容显示在EditText控件中
- 如果onKey()函数返回true，将阻止按键事件的继续传递，这样EditText控件就不能够捕获到按键事件，也就不能够将按键内容显示在EditText控件中

5.6 界面事件

•5.6.1 按键事件

- Android界面框架支持对按键事件的监听，并能够将按键事件的详细信息传递给处理函数
- 为了处理控件的按键事件，先需要设置按键事件的监听器，并重载onKey()函数
- 示例代码如下：

```
1. entryText.setOnKeyListener(new OnKeyListener(){
2. @Override
3. public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
4. //过程代码……
5. return true/false;
6. }
```

5.6 界面事件

•5.6.1 按键事件

- 第1行代码是设置控件的按键事件监听器
- 第3行代码的onKey ()函数中的参数
  - 第1个参数view表示产生按键事件的界面控件
  - 第2个参数keyCode表示按键代码
  - 第3个参数keyEvent则包含了事件的详细信息，如按键的重复次数、硬件编码和按键标志等
- 第5行代码是onKey ()函数的返回值
  - 返回true，阻止事件传递
  - 返回false，允许继续传递按键事件

5.6 界面事件

•5.6.1 按键事件

- KeyEventDemo是说明如何处理按键事件的示例
- KeyEventDemo用户界面
  - 最上方的EditText控件是输入字符的区域
  - 中间的CheckBox控件用来控制onKey()函数的返回值
  - 最下方的TextView控件用来显示按键事件的详细信息
    - 按键动作
    - 按键代码
    - 按键字符
    - Unicode编码
    - 重复次数
    - 功能键状态
    - 硬件编码
    - 按键标志

5.6 界面事件

•5.6.1 按键事件

- 在EditText中，每当任何一个键子按下或抬起时，都会引发按键事件
- 为使EditText处理按键事件，需要使用setOnKeyListener ()函数在代码中设置按键事件监听器，并在onKey()函数添加按键事件的处理过程

```
1. entryText.setOnKeyListener(new OnKeyListener(){
2. @Override
3. public boolean onKey(View view, int keyCode, KeyEvent keyEvent) {
4. int metaState = keyEvent.getMetaState();
5. int unicodeChar = keyEvent.getUnicodeChar();
6. String msg = "";
```

5.6 界面事件

•5.6.1 按键事件

```
7.  msg +="按键动作:" + String.valueOf(keyEvent.getAction())+"\n";
8.  msg +="按键代码:" + String.valueOf(keyCode)+"\n";
9.  msg +="按键字符:" + (char)unicodeChar+"\n";
10. msg +="UNICODE:" + String.valueOf(unicodeChar)+"\n";
11. msg +="重复次数:" + String.valueOf(keyEvent.getRepeatCount())+"\n";
12. msg +="功能键状态:" + String.valueOf(metaState)+"\n";
13. msg +="硬件编码:" + String.valueOf(keyEvent.getScanCode())+"\n";
14. msg +="按键标志:" + String.valueOf(keyEvent.getFlags())+"\n";
15. labelView.setText(msg);
16. if (checkBox.isChecked())
17. return true;
18. else
19. return false;
20. }
```

5.6 界面事件

•5.6.1 按键事件

- 第4行代码用来获取功能键状态。功能键包括左Alt键、右Alt键和Shift键，当这三个功能键被按下时，功能键代码metaState值分别为18、34和65；但没有功能键被按下时，功能键代码metaState值分别为0
- 第5行代码获取了按键的Unicode值，在第9行中，将Unicode转换为字符，显示在TextView中
- 第7行代码获取了按键动作，0表示按下按键，1表示抬起按键。第7行代码获取按键的重复次数，但按键被长时间按下时，则会产生这个属性值
- 第13行代码获取了按键的硬件编码，不同硬件设备的按键硬件编码都不相同，因此该值一般用于调试
- 第14行获取了按键事件的标志符

5.6 界面事件

•5.6.2触摸事件

- Android界面框架支持对触摸事件的监听，并能够将触摸事件的详细信息传递给处理函数
- 需要设置触摸事件的监听器，并重载onTouch ()函数

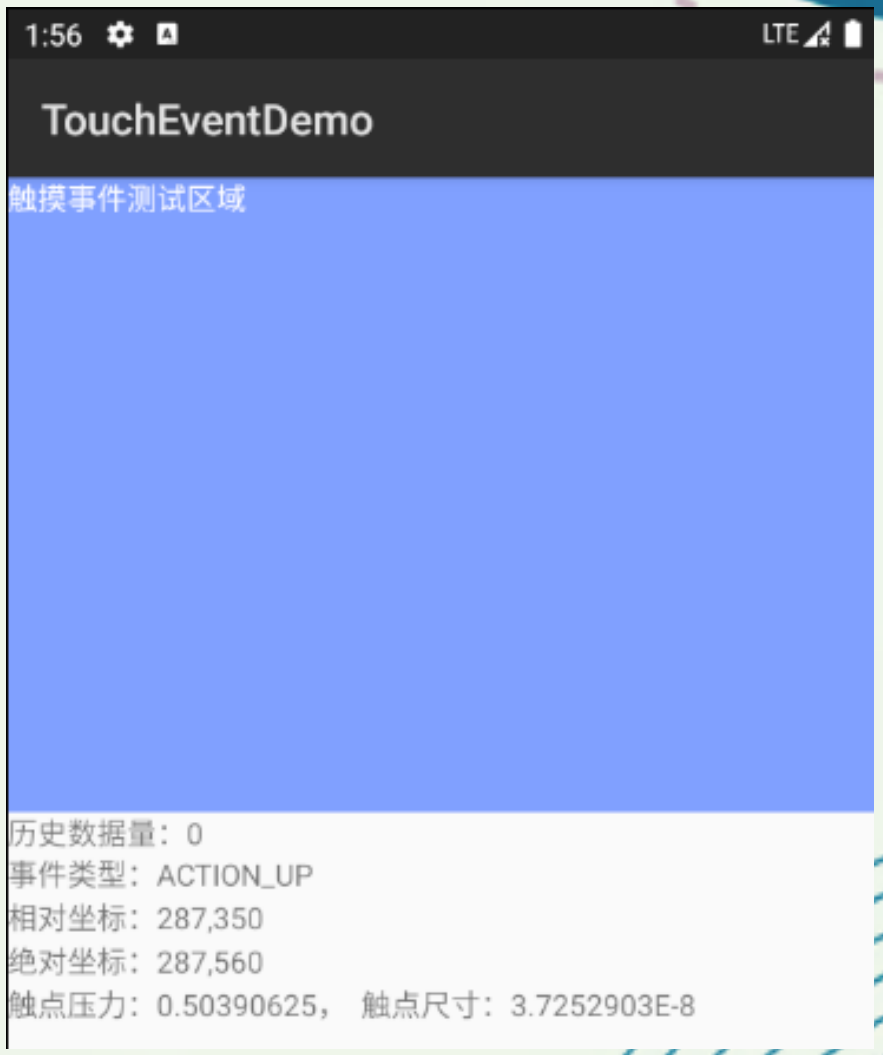
```
1.  touchView.setOnTouchListener(new View.OnTouchListener(){
2.  @Override
3.  public boolean onTouch(View v, MotionEvent event) {
4.  //过程代码……
5.  return true/false;
6.  }
```

  - 第1行代码是设置控件的触摸事件监听器
  - 在代码第3行的onTouch()函数中，第1个参数View表示产生触摸事件的界面控件；第2个参数MontionEvent表示触摸事件的详细信息，如产生时间、坐标和触点压力等
  - 第5行是onTouch()函数的返回值

5.6 界面事件

•5.6.2触摸事件

- TouchEventDemo是一个说明如何处理触摸事件的示例
- TouchEventDemo用户界面
  - 浅蓝色区域是可以接受触摸事件的区域，用户可以在Android模拟器中使用鼠标点击屏幕，用以模拟触摸手机屏幕
  - 下方黑色区域是显示区域，用来显示触摸事件的类型、相对坐标、绝对坐标、触点压力、触点尺寸和历史数据量等信息



5.6 界面事件

•5.6.2触摸事件

- 在用户界面中使用了线性布局，并加入了3个TextView控件
  - 第1个TextView（ID为touch_area）用来标识触摸事件的测试区域
  - 第2个TextView（ID为history_label）用来显示触摸事件的历史数据量
  - 第3个TextView（ID为event_label）用来触摸事件的详细信息，包括类型、相对坐标、绝对坐标、触点压力和触点尺寸

5.6 界面事件

•5.6.2触摸事件

- XML文件的代码如下:

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3.  android:orientation="vertical"
4.  android:layout_width=“match_parent"
5.  android:layout_height=“match_parent">
6.  <TextView  android:id="@+id/touch_area"
7.  android:layout_width=“match_parent"
8.  android:layout_height="300dip"
9.  android:background="#0FF"
10. android:textColor="#FFFFFF"
11. android:text="触摸事件测试区域">
12. </TextView>
```

5.6 界面事件

•5.6.2触摸事件

```
13.   <TextView android:id="@+id/history_label"
14.                         android:layout_width="wrap_content"
15.   android:layout_height="wrap_content"
16.   android:text="历史数据量：" >
17.   </TextView>
18.   <TextView android:id="@+id/event_label"
19.   android:layout_width="wrap_content"
20.   android:layout_height="wrap_content"
21.   android:text="触摸事件：" >
22.   </TextView>
23.    </LinearLayout>
```

- 第9行代码定义了TextView的背景颜色，#80A0FF是颜色代码
- 第10行代码定义了TextView的字体颜色

5.6 界面事件

•5.6.2触摸事件

- 在代码中为了能够引用XML文件中声明的界面元素，使用了下面的代码

```
1. TextView labelView = null;
2. labelView = (TextView)findViewById(R.id.event_label);
3. TextView touchView = (TextView)findViewById(R.id.touch_area);
4. final TextView historyView = (TextView)findViewById(R.id.history_label);
```

5.6 界面事件

•5.6.2触摸事件

- 当手指接触到触摸屏并且在触摸屏上移动或离开触摸屏时，分别会引发ACTION_DOWN、ACTION_UP和ACTION_MOVE触摸事件，而无论是哪种触摸事件，都会调用onTouch()函数进行处理
- 事件类型包含在onTouch()函数的MotionEvent参数中，可以通过getAction()函数获取到触摸事件的类型，然后根据触摸事件的不同类型进行不同的处理
- 为了能够使屏幕最上方的TextView处理触摸事件，需要使用setOnTouchListener()函数在代码中设置触摸事件监听器，并在onTouch()函数添加触摸事件的处理过程

5.6 界面事件

•5.6.2触摸事件

```
1.  touchView.setOnTouchListener(new View.OnTouchListener(){
2.          @Override
3.          public boolean onTouch(View v, MotionEvent event) {
4.              int action = event.getAction();
5.              switch (action) {
6.                case (MotionEvent.ACTION_DOWN):
7.                Display("ACTION_DOWN",event);
8.                break;
9.                case (MotionEvent.ACTION_UP):
10.               int historySize = ProcessHistory(event);
11.               historyView.setText("历史数据量："+historySize);
12.               Display("ACTION_UP",event);
13.               break;
14.               case (MotionEvent.ACTION_MOVE):
15.               Display("ACTION_MOVE",event);
16.               break;
17.             }
```

5.6 界面事件

•5.6.2触摸事件

```
18.                 return true;
19.             }
20.         });
```

- 第7行代码的Display()是一个自定义函数，主要用来显示触摸事件的详细信息，函数的代码和含义将在后面进行介绍
- 第10行代码的ProcessHistory()也是一个自定义函数，用来处理触摸事件的历史数据，后面进行介绍
- 第11行代码是使用TextView显示历史数据的数量

5.6 界面事件

•5.6.2触摸事件

- MotionEvent参数中不仅有触摸事件的类型信息，还触点的坐标信息，获取方法是使用getX()和getY()函数，这两个函数获取到的是触点相对于父界面元素的坐标信息。如果需要获取绝对坐标信息，则可使用getRawX()和getRawY()函数
- 触点压力是一个介于0和1之间的浮点数，用来表示用户对触摸屏施加压力的大小，接近0表示压力较小，接近1表示压力较大，获取触摸事件触点压力的方式是调用getPressure()函数
- 触点尺寸指用户接触触摸屏的接触点大小，也是一个介于0和1之间的浮点数，接近0表示尺寸较小，接近1表示尺寸较大，可以使用getSize()函数获取

5.6 界面事件

•5.6.2触摸事件

- Display()将MotionEvent参数中的事件信息提取出来，并显示在用户界面上

```
1.  private void Display(String eventType, MotionEvent event){
2.      int x = (int)event.getX();
3.      int y = (int)event.getY();
4.      float pressure = event.getPressure();
5.      float size = event.getSize();
6.      int RawX = (int)event.getRawX();
7.      int RawY = (int)event.getRawY();
8.
9.      String msg = "";
10.     msg += "事件类型： " + eventType + "\n";
11.     msg += "相对坐标： "+String.valueOf(x)+","+String.valueOf(y)+"\n";
12.     msg += "绝对坐标： "+String.valueOf(RawX)+","+String.valueOf(RawY)+"\n";
13.     msg += "触点压力： "+String.valueOf(pressure)+"，  ";
14.     msg += "触点尺寸： "+String.valueOf(size)+"\n";
15.     labelView.setText(msg);
16.     }
```

5.6 界面事件

•5.6.2触摸事件

- 一般情况下，若用户将手指放在触摸屏上，但不移动，然后抬起手指，应先后产生ACTION_DOWN和ACTION_UP两个触摸事件
- 但如果用户在屏幕上移动手指，然后再抬起手指，则会产生这样的事件序列：ACTION_DOWN → ACTION_MOVE → ACTION_MOVE → ACTION_MOVE → ……→ ACTION_UP

5.6 界面事件

•5.6.2触摸事件

- 在手机上运行的应用程序，效率是非常重要的。如果Android界面框架不能产生足够多的触摸事件，则应用程序就不能够很精确的描绘触摸屏上的触摸轨迹
- 如果Android界面框架产生了过多的触摸事件，虽然能够满足精度的要求，但却降低了应用程序效率
- Android界面框架使用了“打包”的解决方法。在触点移动速度较快时会产生大量的数据，每经过一定的时间间隔便会产生一个ACTION_MOVE事件，在这个事件中，除了有当前触点的相关信息外，还包含这段时间间隔内触点轨迹的历史数据信息，这样既能够保持精度，又不至于产生过多的触摸事件

5.6 界面事件

•5.6.2触摸事件

- 通常情况下，在ACTION_MOVE的事件处理函数中，都先处理历史数据，然后再处理当前数据

```
1.  private int ProcessHistory(MotionEvent event)
2.      {
3.      int historySize = event.getHistorySize();
4.      for (int i = 0; i < historySize; i++) {
5.      long time = event.getHistoricalEventTime(i);
6.      float pressure = event.getHistoricalPressure(i);
7.      float x = event.getHistoricalX(i);
8.      float y = event.getHistoricalY(i);
9.      float size = event.getHistoricalSize(i);
10.
11.     // 处理过程.....
12.     }
13.     return historySize;
14.     }
```

5.6 界面事件

- **5.6.2触摸事件**
  - 第3行代码获取了历史数据的数量
  - 然后在第4行至12行中循环处理这些历史数据
  - 第5行代码获取了历史事件的发生时间
  - 第6行代码获取历史事件的触点压力
  - 第7行和第8行代码获取历史事件的相对坐标
  - 第9行获取历史事件的触点尺寸
  - 在第14行返回历史数据的数量，主要是用于界面显示
- Android模拟器并不支持触点压力和触点尺寸的模拟，所有触点压力恒为0.50390625
- 同时Android模拟器上也无法产生历史数据，因此历史数据量一直显示为0

习题：

- 1.简述6种界面布局的特点。
- 2.参考下图中界面控件的摆放位置，使用多种布局方法实现用户界面，并对比各种布局实现的复杂程度和对不同屏幕尺寸的适应能力。

- 3.简述Android系统三种菜单的特点及其使用方式。
- 4.说明使用操作栏为程序开发所带来的便利。